Homework 10: bigger files for xv6

제출 마감: 2018. 12. 6. 11:00 am

제출처: 오준택 조교 (na94jun@gmail.com)

본 과제는 xv6에서 사용하는 파일의 최대 크기를 증가시키는 과제이다. 현재 xv6 파일의 최대 크기는 71,680 바이트로, 블록 140개에 해당된다. 이는 xv6의 inode가 12개의 “direct” 블록 포인터와 하나의 “singly-indirect” 블록 포인터를 갖고 있어 최대 12+128=140 개의 블록을 가리킬 수 있기 때문이다. 본 과제에서는 xv6 파일 시스템이 “doubly-indirect” 블록을 지원하도록 변경할 것이다. “doubly-indirect” 블록은 128개의 “singly-indirect” 블록 포인터를 갖는다. “singly-indirect” 블록은 128개의 데이터 블록 포인터를 갖는다. 즉, “doubly-indirect” 블록 하나는 128*128=16384개의 데이터 블록을 가리킬 수 있다. 본 과제 수행 후, inode는 11개의 “direct” 블록 포인터와 1개의 “singly-indirect” 블록 포인터와 1개의 “doubly-indirect” 블록 포인터를 갖게 될 것이고, 총 11+128+16384=16523개의 블록을 가리킬 수 있게 될 것이다. 즉, 파일의 최대 크기는 16523*512=약 8.5 megabytes로 확장될 것이다.

준비 사항

다음과 같이 Makefile을 수정하여 가상 머신의 CPU 개수를 1개로 제한하시오.

```
ifndef CPUS
CPUS := 1
endif
```

그리고, 다음과 같은 코드를 QEMUOPTS 설정 코드 위에 추가하시오.

```
QEMUEXTRA = -snapshot
QEMUOPTS = -drive file=fs.img,index=1,media=disk,format=raw
-drive file=xv6.img,index=0,media=disk,format=raw -smp $(CPUS)
-m 512 $(QEMUEXTRA)
```

위와 같이 Makefile을 수정하면 xv6가 large file을 생성할 때의 속도가 크게 빨라진다.

현재 mkfs는 파일 시스템이 1000개 이하의 데이터 블록을 갖도록 초기화한다. 이는 과제를 수행하기에 매우 부족한 양이다. param.h 파일을

다음과 같이 수정하여 파일 시스템이 갖을 수 있는 데이터 블록의 개수를 증가시키시오.

```
#define FSSIZE       20000  // size of file system in blocks
```

piazza에서 big.c 파일을 다운로드 받아 xv6 디렉토리에 저장하고 UPROGS 리스트에 "_big"을 추가한 후, xv6에서 big 을 실행시키시오. big은 xv6가 허용하는 최대 크기를 갖는 파일을 생성하고 그 크기를 출력한다. 과제 수행 전에는 140 sectors라고 출력되어야 한다.

필수 사전 지식

디스크에서 파일의 데이터를 찾는 함수는 fs.c에 구현되어 있는 bmap()이다. 코드를 보고 이해한 후 과제에 임하기 바란다. bmap()은 파일을 읽거나 쓸 때 호출된다. 파일에 데이터를 쓸 때는, bmap()은 새로운 블록을 할당하여 파일의 내용을 저장할 수 있다. 또한, 파일의 크기가 너무 클 경우 indirect block을 할당받아 더 많은 데이터 블록 번호를 저장할 수 있도록 할 수도 있다.

bmap()은 두 종류의 블록 번호를 사용한다. bmap()의 인자 중 bn이 있다. bn은 "logical block"이며, 해당 블록 번호는 파일의 시작부터 인덱싱한 번호다. inode에 있는 ip->addrs[]와 bread()의 인자는 디스크 블록 번호이다. bmap()은 파일의 logical block 번호를 디스크 블록 번호로 바꾸는 함수라고 할 수 있다.

해야 할 일

bmap()이 doubly-indirect block을 추가 지원하도록 수정하시오. direct block의 개수를 12개에서 11개로 하나 줄여서 doubly-indirect block 포인터를 위한 공간을 마련해야 할 것이다. 반드시 on-disk inode의 크기는 변경되면 안된다. ip-addrs[]에서 첫번째부터 11개의 블록 포인터들은 direct block으로 지정하고, 12번째 블록 포인터는 singly-indirect block 포인터로 지정하고 마지막 13번째 블록 포인터는 doubly-indirect block 포인터로 지정하시오.

또한, bmap() 함수를 수정하여 doubly-indirect block를 지원하도록 하시오. (파일을 삭제하는 경우는 고려하지 않아도 된다.)

과제를 완벽히 수행하고 나면, big 프로그램은 16523 sectors를 출력할 것이다. 이를 끝내기까지 수십초가 걸릴 수도 있다.

힌트

bmap()을 완전히 이해하고 과제를 수행하시오. ip-addrs[], indirect block, doubly-indirect block, singly-indirect block, data block들의 관계를 나타내는 다이어그램을 그려보면 이해가 잘 될 것이다. 또한, doubly-indirect block을 추가하는 것이 왜 파일의 최대 크기를 16384 sector 만큼 증가시키는지 이해한 후 과제를 진행하시오.

doubly-indirect과 indirect block이 가리키는 블록들의 logical block 번호를 어떻게 구해야 하는지 생각해보시오.

NDIRECT가 바뀌면, file.h에 선언된 struct inode의 addrs[]의 크기를 바꿔야 할 것이다. struct inode의 addrs와 struct dinode의 addrs가 갖은 크기를 갖도록 해야 한다.

mkfs도 파일 시스템을 포맷하기 위해서 NDIRECT를 사용한다. 그래서 NDIRECT를 바꾸면 fs.img를 새로 만들어야 한다. fs.img는 지우고 make하면 다시 생성된다. 다시 생성하시오.

만약 크래시가 발생해서 파일 시스템에 이상이 생기면 fs.img를 지우고 다시 생성하시오.

indirect block과 doubly-indirect block은 필요할 때만 할당된다. 기존 bmap()은 이미 indirect block을 필요할 때만 할당하고 있다. 이를 참조하시오.

Submit: 수정한 fs.c

제출 양식: hw10_학번.c